AMENDMENTS TO THE SPECIFICATION:

Please delete the paragraph beginning at page 1, line 19 and replace with the following paragraph:

The devices 14 and 16 can be any network devices capable of transmitting and/or receiving network traffic data, such as framing/ Media Access Control (MAC) ~~MAC~~ devices, e.g., for connecting to 10/100BaseT Ethernet, Gigabit Ethernet, Asynchronous Transfer Mode (ATM) ~~ATM~~ or other types of networks, or devices for connecting to a switch fabric. For example, in one arrangement, the network device 14 could be an Ethernet MAC device (connected to an Ethernet network, not shown) that transmits data to the processor 12 and device 16 could be a switch fabric device that receives processed data from processor 12 for transmission onto a switch fabric. In such an implementation, that is, when handling traffic to be sent to a switch fabric, the processor 12 would be acting as an ingress network processor. Alternatively, the processor 12 could operate as an egress network processor, handling traffic that is received from a switch fabric (via device 16) and destined for another network device such as network device 14, or network coupled to such device.

Please delete the paragraph beginning at page 20, line 12 and replace with the following paragraph:

The thread execution order is controlled via the inter-thread signaling. The next thread signal may be given by writing to the SAME_ME_SIGNAL CSR 82 (from FIG. 3). As shown in ~~the~~ a table 120 of FIG. 6, a write to the SAME_ME_SIGNAL CSR 82 requires a write latency

122, a read latency 124 and a usage latency 126, specified in terms of instruction cycles. The write latency 122 specifies the minimum number of instruction cycles "j" from the write instruction ("local_csr_wr[]") to the time that the CSR is actually written. The write latency, shown in the table as having a minimum value of j=3, is of particular importance with respect to the thread execution order, as will be explained more fully below.

Please delete the paragraph beginning at page 26, line 18 and replace with the following paragraph:

Referring back to phase processing 166, if the executed phase processing is completed for phase 2, it is determined 178 if the thread is the last thread. If the thread is not the last thread, the thread signals 180 the next thread m+1. If the thread is the last thread, the thread instead signals 182 the first thread of the next ME. After either signaling 180 or 182, the thread performs a context swap 184, thus allowing the next thread, thread 0, on the next ME to begin execution of the critical section.

Please delete the paragraph beginning at page 27, line 3 and replace with the following paragraph:

FIGS. 10A and 10B illustrate the impact of the CSR write latency on thread execution. As shown in FIG. 10A, if a currently executing thread, indicated in the example as thread 3 190, executes the instruction 2 (which causes it to wait for the next thread signal from previous thread 2) in a number of cycles k after instruction 1 (which causes thread 3 190 to signal thread 4 194) where k is less than the write latency, e.g., 3 cycles, then a thread other than thread 4 194 may begin executing after thread 3 190. In the illustrated example, thread execution order is lost, as thread 0 192 follows thread 3 190. In contrast, FIG. 10B shows that thread order execution is maintained, that is, thread 4 194 follows thread 3 190, when k is at least 3 cycles.

Please delete the paragraph beginning at page 28, line 15 and replace with the following paragraph:

As shown in FIG. 11A, ~~the~~ threads 210a on ME 0 execute out of order, as threads 0 and 1 follow thread 3 and execute before order is returned with the execution of threads 4 though 7. FIG. 11B shows ~~the~~ threads on ME 0 210b executing in sequential order ~~on ME 0~~. Because ME 1 having threads 211 receives an inter-ME signal 212 from ME 0 earlier under the optimized conditions depicted in FIG. 11B (compared to those depicted in FIG. 11A), it can be seen that ME 1 requires fewer idle cycles when thread execution order is maintained.

Please delete the paragraph beginning at page 29, line 1 and replace with the following paragraph:

Such is also the case when context switching occurs, as illustrated in FIGS. 12A and 12B. In FIG. 12A, threads on ME 0 310a proceed with the execution of other code, e.g., some non-critical section code, immediately following the completion of the critical section processing without performing a context swap, thus delaying the inter-thread signaling to the next threads and ME enabling those threads and ME to begin execution of the critical section. In FIG. 12B, the threads on ~~ME0~~ ME 0 310b promptly swap out to allow the next thread to execute, which means that thread 7 ends up signaling 312 to next ME (ME 1 having threads 311) at an earlier time than was the case under the circumstances depicted in FIG. 12A. Referring to both figures, it can be seen that the delay caused by the lack of context swap (shown in FIG. 12A) results in an increase in the number of idle cycles on the ME 1 as it waits to execute the critical section.